Services Web : du test actif au test passif

Tien-Dung Cao, Richard Castanet, Patrick Félix

Services Web : du test actif au test passif

Tien-Dung Cao[1], **Richard Castanet**[2], **Patrick Felix**[2]

[1]*Département d'Informatique - Université de Tan Tao*
Tan Duc E-city, District de Duc Hoa, Long An, Vietnam.
dung.cao@ttu.edu.vn

[2]*Université de Bordeaux - LaBRI, UMR CNRS 5800*
351 crs de la Libération, 33405 Talence, France.
{castanet,felix}@labri.fr

RÉSUMÉ. *Nous présentons dans cet article deux approches pour effectuer des tests pour les services Web. Le test actif OnLine permet le test unitaire d'une orchestration de services web et est mis en oeuvre par l'outil WSOTF. Pour le test passif, la méthodologie proposée peut être utilisée soit pour vérifier une trace (vérification off-line), soit pour la vérification d'exécution (vérification on-line) avec des contraintes de temps. Afin de réaliser cela, nous utilisons dans un premier temps le langage Nomad pour définir les règles de vérification. Puis, nous proposons un algorithme qui permet de vérifier simultanément plusieurs instances de règles. En plus du cadre théorique, nous avons développé un outil logiciel, appelé RV4WS (Runtime Verification engine for Web Service), qui aide à l'automatisation de notre approche de test passif. L'algorithme présenté dans cet article est mis en œuvre dans l'outil. Nous présentons aussi un mécanisme pour collecter les traces observées.*

ABSTRACT. *We present here two approaches for testing for Web services. Active testing allows the unit test an orchestration of web services and is implemented by the tool WSOTF. For passive testing, the proposed methodology can be used either to check a trace (offline checking) or to runtime verification (online checking) with timing constraints, including future and past time. In order to perform this: firstly, we use the Nomad language to define the verification rules. Secondly, we propose an algorithm that can check simultaneously multi instances. Afterwards, with each verification rule, we propose a graphical statistics, with some fixed properties, that helps the tester to easy assess about the service. In addition to the theoretical framework we have developed a software tool, called RV4WS (Runtime Verification engine for Web Service), that helps in the automation of our passive testing approach. In particular the algorithm presented in this paper is fully implemented in the tool. We also present a mechanism to collect the observable trace.*

MOTS-CLÉS : *Langage Nomad, Service Web, Spécification, Test actif, Test Passif, Vérification d'exécution.*

KEY WORDS: *Active testing, Nomad language, Passive testing, Runtime verification, Rule specification, Web service.*

1. Introduction

Les architectures orientées services (SOA) constituent un paradigme architectural qui a acquis une grande popularité ces dernières années. L'idée principale de SOA consiste à réutiliser et intégrer les services faiblement couplés et surtout distribués. L'interface d'un service est spécifiée au moyen d'un langage de description d'interface ce qui permet d'invoquer un service sans en connaître sa mise en œuvre. De nos jours, les services Web sont des applications qui permettent la construction d'applications SOA en composant d'autres services web. Il existe deux types de composition qu'il est important de distinguer :

– **Orchestration** : une orchestration expose la logique interne d'un seul composant en spécifiant la structure de contrôle de flux et les dépendances de flux de données de cet élément. Dans une orchestration, il n'y a qu'un seul processus principal (nommé chef d'orchestre) qui contrôle ses services partenaires en utilisant une architecture centrale.

– **Chorégraphie** : une chorégraphie décrit une collaboration de services dans un service composite. Dans ce contexte, il n'y a pas de processus principal et tous participent à la composition et interagissent les uns avec les autres.

Deux approches classiques peuvent être utilisées lors du développement pour tester un service web.

– **Test Actif** : dans une composition de services web, la mauvaise qualité d'un service peut affecter d'autres services partenaires et leur composition. Nous appliquerons une technique de test actif pour trouver les erreurs de chaque service de façon isolée sans l'interaction de ses partenaires, mais uniquement en les simulant. Nous sommes donc au niveau du test unitaire. Nous nous concentrons ici sur les tests de conformité de composition de services web. En outre, les contraintes de temps doivent être prises en compte dans notre approche de test puisque les services Web sont des applications temps réel.

– **Test passif** : c'est une approche appliquée sur les services en fonctionnement pour vérifier certaines propriétés. Trouver tous les défauts des services au moyen de tests est impossible, donc les surveiller après la publication est une pratique qui peut s'avérer intéressante. Dans de nombreux cas, nous ne pouvons pas appliquer une méthode de test actif pour tester certains systèmes en cours d'exécution. Le test passif est une technique de surveillance de certaines propriétés des services : pour donner le verdict (true/false) on collecte les traces d'exécution et on les analyse. Nous nous sommes intéressés à utiliser cette approche pour tester un service Web après sa publication.

Dans le reste de cet article, nous discutons, section 2, des travaux connexes et présentons, sections 3 et 4, des outils pris en charge qui peuvent s'appliquer dans les différentes phases de test des services Web. Finalement, la dernière partie de notre article présente une conclusion et des perspectives (section 5).

2. Travaux connexes

Au cours de ces dernières années, de nombreuses méthodes et outils sont proposés et développés pour le test passif d'un service web (y compris une composition de services web). Ces travaux se

sont focalisés soit sur la vérification d'une trace afin de rendre un verdict, soit sur des méthodes statistiques et dynamiques de certaines propriétés des services web.

[DRA 09] propose l'utilisation de 'Stream X-machines' pour la construction de spécifications formelles du comportement des services Web. Les auteurs présentent également un monitoring de l'exécution mais ne présentent pas un outil pour vérifier une trace d'exécution en utilisant la spécification de 'Stream X-machines' des services web.

[BAR 09] présente un cadre de suivi pour l'orchestration BPEL. Leur travail porte surtout sur les caractéristiques du comportement de la composition exprimée dans BPEL plutôt que sur les services Web. Cependant, une évaluation (un verdict $true/false$) sur le service n'est pas abordé dans ce travail.

[CAV 10] propose un mécanisme de collecte de traces pour SOA en intégrant des modules dans le moteur BPEL et un outil vérifie en «off-line» les traces d'exécution. Cette approche utilise aussi le langage Nomad pour définir les règles de sécurité. Mais il ne nous permet pas de vérifier les propriétés en temps réel («on-line»).

Les travaux de Li et al ([LI 05], [LI 06]) présentent des opérateurs pour définir des contraintes d'interaction entre services Web. Les auteurs utilisent les automates à états finis comme modèle et le processus de validation se déroule en même temps que la collecte des traces. Cependant ce travail ne tient pas compte des contraintes de temps des services web.

3. Test actif OnLine de services web : l'outil WSOTF

Le test actif est une approche où le testeur envoie une requête à l'implémentation sous test (IST), recueille les réponses, les analyse afin de produire un verdict. Nous présentons ici une approche de test actif $OnLine$ pour le test unitaire d'une orchestration de services web.

Le test on-line est une approche dans laquelle la génération de test et l'exécution des tests sont exécutées en parallèle en sélectionnant les interactions du cas de test au hasard. Dans l'état actuel des choses, cette approche ne prend pas en compte des critères de couverture ([CAO 10B]).

WSOTF (Web Service Online Testing Framework) offre une plateforme on-line ([CAO 10B]) qui permet de générer les cas de test pour les services Web et de les exécuter. Il peut être utiliser pour le test unitaire d'un service web (Spéc. WSDL) ou une orchestration de services web. Il est mis en oeuvre en Java et son architecture détaillée apparaît dans la figure 1. Il se compose de deux parties : le contrôleur et l'adaptateur. Le contrôleur est composé de cinq éléments principaux : un chargeur, un traitement des données, une bibliothèque de fonctions données, une interface pour envoyer/recevoir des messages vers/de IST et un exécuteur.

L'entrée de cet outil est la spécification du service : c'est un automate temporisé étendu sous forme d'un fichier XML (qui peut être obtenu à partir de la description BPEL) qui comprend une section 'partenaires' déclarant le nom et l'emplacement des spécifications WSDL des partenaires, la liste des variables (les types de variables sont : int, $boolen$ ou un type de message de SOAP défini dans le WSDL), l'horloge locale, l'état initial, et une liste de transitions. Chaque transition est composée de sept champs : état source, état cible, événement, garde sur les variables, la garde sur

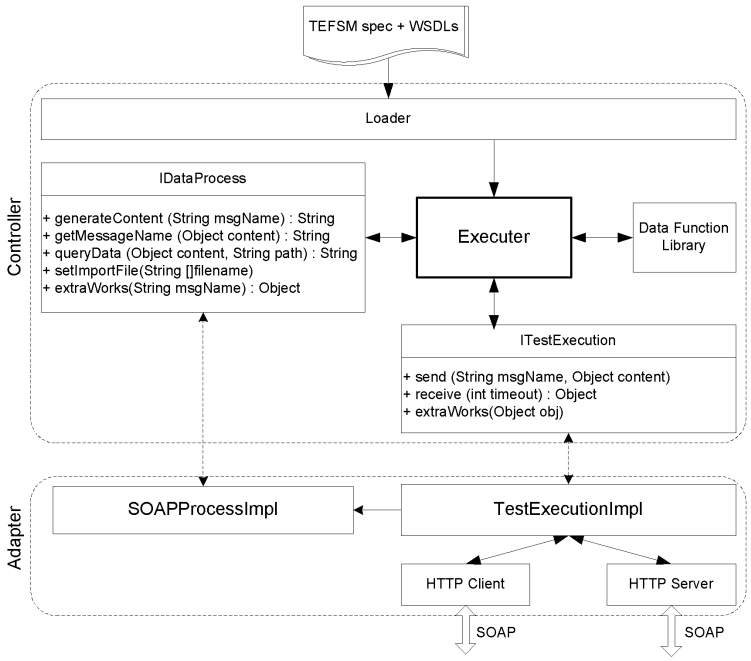

Figure 1. *Architecture de la plateforme WSOTF*

l'horloge, les données mises à jour et la remise à zéro horloge locale. En WSOTF, nous utilisons la forme *?pl.pt.op.msg* pour représenter une action d'entrée qui signifie la réception du message *msg* pour l'opérateur *op* de la portTyte *pt* du partenaire *pl* et la forme *!pl.pt.op.msg* pour représenter une action de sortie. Le résultat de WSOTF (les traces, y compris l'intervalle de temps entre deux actions et son verdict correspondant) est sauvegardé dans un fichier xml.

4. Test passif de services web : l'outil RV4WS

Pour le test passif, on peut citer les approches $OnLine$ et $OffLine$. L'approche $OnLine$ vérifie une trace d'exécution d'un service web en même temps que son exécution.

L'outil RV4WS (Runtime Verification engine for Web Service) concerne le test passif ($OnLine$ et $OffLine$) en vérifiant une trace temporelle par rapport un ensemble de propriétés (règle de contraintes). Ces propriétés permettent de prendre en compte les contraintes de temps, des conditions sur le contenu des messages, des corrélations de données, etc.

Ces règles seront exprimées en langage Nomad [CUP 05] habituellement utilisé pour des propriétés de sécurité. Nous avons choisi ce langage parce qu'il fournit un moyen de spécifier les actions qui doivent apparaître (appelées autorisation ou obligation dans le langage Nomad) ou qui ne doivent pas apparaître (appelées interdiction dans le langage Nomad).

Ce langage a déjà été utilisé pour définir les règles de sécurité dans l'approche de [CAV 09]. Nous proposons un algorithme, mis en œuvre dans l'outil RV4WS ([CAO 10], [CAO 10a]) , qui vérifie en $OnLine$ ou $OffLine$, y compris le temps futur et passé, une séquence d'événements d'entrée/sortie.

Une méthode de test passif est composée de trois étapes : 1) définir l'architecture de test passif pour recueillir les traces d'exécution 2) définir les règles de contrainte (nous utilisons le langage Nomad pour ce faire) et 3) analyser ($OnLine$ ou $OffLine$) les traces d'exécution pour donner le verdict. Ce verdict est $PASS$ si le système respecte les règles de contraintes spécifiées et $FAIL$ dans le cas contraire. Le verdict $INCONCLUSIVE$ est possible en cas de contrôle $OffLine$ avec des informations incomplètes.

Nous proposons deux architectures pour la collecte de traces avec la notion de point d'observation (PO), une pour le service Web et une autre pour la composition de services web. Notre PO permettra de recueillir la trace d'exécution au niveau du réseau. Avec un service web, nous avons un PO entre le client et le service pour recueillir les messages SOAP. Avec une composition de services web, nous avons un PO entre le service Web et chaque partenaire (figure 2). Tous les messages collectés par les PO seront envoyés à un moteur pour analyse et génération de verdict ([CAO 10] et [CA0 10A]).

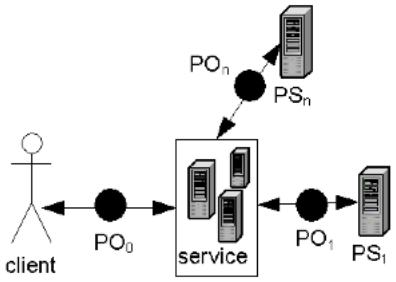

Figure 2. *Architecture pour la composition de services web*

Afin de visualiser les résultats, nous affichons graphiquement des propriétés statistiques sur le temps (temps min, max et moyen) pour chaque règle.

5. Conclusion et travaux futurs

Il existe deux approches pour tester un système : le test actif où un testeur interagit directement avec l'IST et vérifie la correction des réponses pour donner le verdict, et le test passif qui recueille des traces d'exécution et les analyse pour donner le verdict. Cette étude se concentre sur le test passif et actif des services Web. Nous avons proposé d'utiliser le langage Nomad pour définir les règles à vérifier et un algorithme pour vérifier la correction d'une trace d'exécution. Cet algorithme a été pleinement mis en œuvre dans l'outil RV4WS avec deux approches : la vérification $OnLine$ et $OffLine$.

Deux poursuites sont envisagées pour ce travail. La première consiste à combiner ces deux approches au sein d'un même étude de cas : on utilise WSOTF (test actif) pour lancer le test actif, et à l'aide de proxy on collecte des traces qui seront analysées par RF4WS (test passif). La version actuelle ne supporte pas encore la corrélation de données et nous souhaitons donc étendre l'outil RV4WS pour éviter cette restriction. La deuxième consiste à étendre le test passif vers les règles de sécurité.

6. Remerciements

Cette recherche est soutenue par l'Agence Nationale française de Recherche au sein du projet WebMov `http://webmov.lri.fr`

7. Bibliographie

[BAR 09] BARTOLINI C., BERTOLINO A., MARCHETTI E., POLINI A., « WS-TAXI : A WSDL-based Testing Tool for Web Services », *International Conference on Software Testing Verification andValidation*, Denver, Colorado - USA, 01-04 avril2009, p. 326-335.

[CAO 10] CAO D., PHAN-QUANG T., FELIX P., CASTANET R., « Automated Runtime Verification for Web Services », *The IEEE International Conference on Web Services*, Miami, Florida, USA, 2010, p. 76-82.

[CAO 10a] CAO D., FELIX P., CASTANET R., « WSOTF : An automatic testing tool for web services composition », *The Fifth International Conference on Internet and Web Applications and Services*, Barcelona, Spain, 2010, p. 7-12.

[CAO 10b] CAO D., « Test and Validation of Web services », Thèse de doctorat, Université de Bordeaux 1, École Doctorale de Mathématique et d'Informatique, 2010.

[CAV 09] CAVALLI A., BENAMEUR A., MALLOULI W., LI K., « A Passive Testing Approach for Security Checking and its Pratical Usage for Web Services Monitoring », *NOTERE 2009*, Montreal, Canada, 2009.

[CAV 10] CAVALLI A., CAO D., MALLOULI W., MARTINS E., SADOVYKH A.SALVA S., ZAIDI F., « WebMov : An dedicated framework for the modelling and testing of Web services composition », *The IEEE International Conference on Web Services*, Miami, Florida, USA, 2010, p. 377-384.

[CUP 05] CUPPENS F., CUPPENS-BOULAHIA N., SANS T., « Nomad : A Security Model with Non Atomic Actions and Deadlines », *CSFW, 2005*, p. 186-196.

[DRA 09] DRANIDIS D., RAMOLLARI E., KOURTESIS D., « Run-time Verification of Behavioural Conformance for Conversational Web Services », *Seventh IEEE European Conference on Web Services*, Eindhoven, The Netherlands, 2009, p. 139-147.

[LAL 08] LALLALI M., ZAIDI F., CAVALLI A., HWANG I., « Automatic Timed Test Case Generation for Web Services Composition », *European Conference on Web Services (ECOWS'08)*, Dublin, Ireland, 2008.

[LI 05] LI Z. , SUN W. , JIANG Z. , ZHANG X., « BPEL4WS Unit Testing : Framework and Implementation », *IEEE International Conference on Web Service*, Orlando, FL, USA, 2005, p. 103-110.

[LI 06] LI Z. , YAN J. , HAN J., « A Runtime Monitoring and Validation Framework for Web Service Interactions », *The Australian Software Engineering Conference ASWEC'06*, 2006, p. 70-79.